Computational Geometry 43 (2010) 410–418

Contents lists available at ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Computing the visibility map of fat objects ☆

Mark de Berg, Chris Gray *,[1]

Department of Computing Science, TU Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Received 2 October 2007
Received in revised form 5 September 2008
Accepted 8 December 2008
Available online 21 June 2009

Keywords:
Fat objects
Realistic input
Visibility map
Hidden-surface removal

ABSTRACT

We give an output-sensitive algorithm for computing the visibility map of a set of n constant-complexity convex fat polyhedra or curved objects in 3-space. Our algorithm runs in $O((n+k)\,\text{polylog}\,n)$ time, where k is the combinatorial complexity of the visibility map. This is the first algorithm for computing the visibility map of fat objects that does not require a depth order on the objects and is faster than the best known algorithm for general objects. It is also the first output-sensitive algorithm for curved objects that does not require a depth order.

1. Introduction

Hidden-surface removal is an important and well-studied computational-geometry problem with obvious applications in computer graphics. The problem is to find those portions of objects in a scene that are visible from a given viewpoint. There are two main approaches to the hidden-surface removal problem: the *image-space approach* and the *object-space approach*. In the former, one calculates the visible object for each pixel of the image; the well-known Z-buffer algorithm is the standard example of this. In the latter, one computes the so-called *visibility map* of the scene, which gives an exact description of the visible part of each object; this is the approach taken in computational geometry.

Formally, the visibility map of a set $\mathcal{P}$ of objects in $\mathbb{R}^3$ with respect to a viewpoint p is defined as the subdivision of the viewing plane into maximal regions such that in each region a single object in $\mathcal{P}$ is visible from p, or no object is visible. We will assume in this paper, as is usual, that the objects are disjoint. The visibility map of a set of n constant-complexity objects can be computed in $O(n^2)$ time [18]. Since the (combinatorial) complexity of the visibility map can be $\Omega(n^2)$—a set of n long and thin triangles that form a grid-like pattern when projected on the viewing plane is an example—this is optimal in the worst case. In most cases, however, the complexity of the visibility map is much smaller than quadratic. Therefore the main challenge in the design of algorithms for computing visibility maps has been to obtain *output-sensitive* algorithms: algorithms whose running time depends not only on the complexity of the input, n, but also on the complexity of the output (that is, the visibility map), k. Ideally the running time should be near-linear in n and k.

The first output-sensitive algorithms for computing visibility maps only worked for polygons parallel to the viewing plane or for the slightly more general case that a depth order on the objects exists and is given [11,14,15,20–22]. Unfortunately a depth order need not exist since there can be cyclic overlap among the objects—see Fig. 1(ii). De Berg and Overmars [8] (see also [3]) developed a method to obtain an output-sensitive algorithm that does not need a depth order. When applied to axis-parallel boxes (or, more generally, c-oriented polyhedra) it runs in $O((n+k)\log n)$ time [8] and when applied to

☆ This research was supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.
* Corresponding author.
E-mail addresses: mdberg@win.tue.nl (M. de Berg), cgray@win.tue.nl (C. Gray).
[1] Current address: Department of Computer Science, TU Braunschweig, Germany.

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2008.12.010

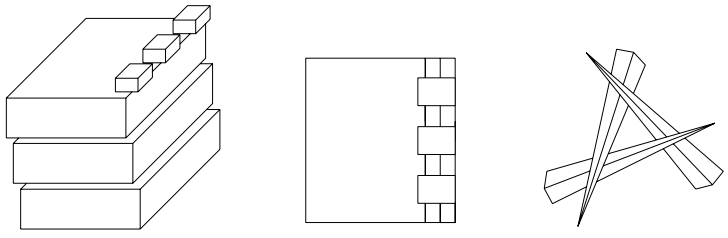

Fig. 1. (i) The visibility map of fat boxes can have quadratic complexity. Left: the scene. Right: the visibility map for $p = (0, 0, \infty)$. (ii) The visibility map of a scene with cyclic overlap.

arbitrary triangles it runs in $O(n^{1+\varepsilon} + n^{2/3+\varepsilon}k^{2/3})$ time [1]. Unfortunately, the running time for the algorithm when applied to arbitrary triangles is not near-linear in n and k; for example, when $k = n$ the running time is $O(n^{4/3+\varepsilon})$. For general curved objects no output-sensitive algorithm is known,[2] not even when a depth order exists and is given.

In this paper we study the hidden-surface removal problem for so-called *fat objects*—see the next section for a definition of fatness. As illustrated in Fig. 1, the complexity of the visibility map of fat objects can still be $\Theta(n^2)$, so also here the main challenge is to obtain an output-sensitive algorithm. Fat objects have received ample attention over the past decade or so, both from a combinatorial and from an algorithmic point of view, and many problems can be solved much more efficiently for fat objects than for general objects. Since hidden-surface removal has been widely studied in computational geometry, it is not surprising that it has also been studied for fat objects: Katz et al. [16] gave an algorithm with running time $O((U(n) + k)\log^2 n)$, where $U(m)$ denotes the maximum complexity of the union of the projection onto the viewing plane of any subset of m objects. Since $U(m) = O(m \log\log m)$ for fat polyhedra [19] and $U(m) = O(\lambda_{s+2}(m)\log^2 m)$ for fat curved objects [5], their algorithm is near-linear in n and k. (Here $\lambda_{s+2}(n)$ is the maximum length of an $(n, s+2)$ Davenport–Schinzel sequence; $\lambda_{s+2}(n)$ is almost linear in n.) However, the algorithm only works if a depth order exists and is given. This leads to the main question we wish to answer: is it possible to obtain an output-sensitive hidden-surface removal algorithm for fat objects that is near-linear in n and k and does not need a depth order on the objects? We answer this question affirmatively by giving an algorithm with running time $O((n+k)$ polylog $n)$ for fat convex objects of constant-complexity. More precisely, the running time is $O((n \log n(\log\log n)^2 + k)\log^3 n)$ when the objects are polyhedra, and it is $O((n\log^{5+\varepsilon} n + k)\log^3 n)$ when the objects are curved.

The only known method [3,8] for output-sensitive hidden-surface removal that can handle objects without depth order needs an auxiliary data structure for ray shooting in so-called *curtains*—these are semi-infinite surfaces, extending downward from the edges of the input objects—and it appears to be difficult to profit from the fact that the objects are fat when implementing this data structure. This also explains why there is currently no efficient output-sensitive algorithm for hidden-surface removal in curved objects: there are no efficient data structures known for ray shooting (with curved rays, in this case) in curved curtains.

Although our algorithm borrows some ideas from the above method—we describe the necessary preliminaries in Section 2—we proceed differently. Instead of building a data structure for ray shooting in curtains in 3D, we project the rays and the objects onto planes "in between" the objects and the rays. Then ray shooting boils down to tracing the rays on these planes in a manner similar to the line-segment-intersection algorithm of Bentley and Ottmann [2]. To make this work, we need a collection of planes such that for every ray and object one of the planes separates them. For this we use a binary space partition on the objects. Our approach works for all convex objects. However, the fatness of the objects is essential to get an efficient algorithm for two main reasons. First, the binary space partition that we use has linear size when applied to fat objects [6]. No such BSP exists for arbitrary objects—every BSP has quadratic size in the worst case when applied to arbitrary objects. The algorithm also computes the union of the projection of the input on the planes "in between" the objects and the rays. The complexity of this union is considerably lower [5,19] for fat objects than for arbitrary objects. Section 3 describes and analyzes our new algorithm in detail. We conclude the paper in Section 4 by mentioning some open problems.

2. Preliminaries

Fatness and low density. Let $\mathcal{P}$ be a set of disjoint convex objects in $\mathbb{R}^3$. We assume the objects are β-fat according to the following definition of fatness [10]: an object o in $\mathbb{R}^d$ is *β-fat* if for any ball b whose center lies in o and that does not fully contain o, we have $\mathrm{vol}(b \cap o) \geqslant \beta \cdot \mathrm{vol}(b)$, where $\mathrm{vol}(o)$ denotes the volume of o. (For convex objects this definition is equivalent, up to constant factors, to other definitions of fatness that have been proposed.)

[2] Some of the algorithms can be generalized to curved objects using standard techniques. The resulting algorithms are not very efficient, however, and typically have running time close to quadratic even when the visibility map has linear complexity.

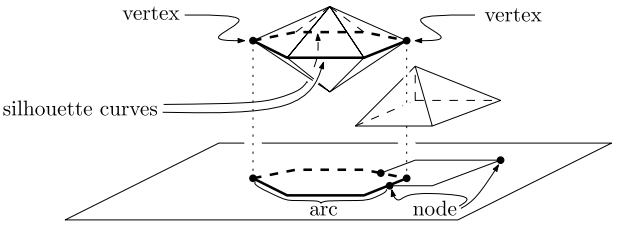

Fig. 2. A scene consisting of two polyhedral objects, and their visibility map. For one of the objects, its silhouette curves and vertices are indicated in bold. One arc and two nodes of the visibility map are indicated explicitly, but in total the visibility map has six arcs and five nodes.

We define size(o), the *size* of an object o, to be the radius of the smallest enclosing ball of o. The *density* of a set S of objects is defined as the smallest number λ such that any ball b is intersected by at most λ objects $o \in S$ such that size(o) $\geqslant$ size(b). The following well-known lemma [10] relates the density of a set of disjoint objects to their fatness.

Lemma 1. *(See [10].) Any set of disjoint β-fat objects has density λ for some $\lambda = O(1/\beta)$.*

Visibility maps. Next we define some notation and terminology relating to visibility maps. We assume from now on that we are looking at the scene from above with the viewpoint at $z = \infty$; hence, we are dealing with a parallel view. As already mentioned, the visibility map $\mathcal{M}(\mathcal{P})$ of a set $\mathcal{P}$ of objects is the subdivision of the viewing plane into maximal regions such that in each region a single object in $\mathcal{P}$ is visible from the viewpoint p, or no object is visible. We assume without loss of generality that the viewing plane is the xy-plane.

Consider an object $o \in \mathcal{P}$. We denote the projection of o onto the viewing plane by proj(o). Since o is convex, the boundary of proj(o) consists of the projection of all points of vertical tangency of o. Let $\sigma(o)$ denote the curve[3] on the boundary of o that projects onto the boundary of proj(o). Note that if o is polyhedral, $\sigma(o)$ consists of certain edges of o. We cut $\sigma(o)$ into two pieces at the points of minimum and maximum x-coordinate; we can assume without loss of generality that these points are unique. We call these pieces *silhouette curves*. Note that for polyhedral objects a silhouette curve consists of multiple edges of the object—see Fig. 2. The endpoints of the silhouette curves are called *vertices*.

$\mathcal{M}(\mathcal{P})$ is a plane graph whose *nodes* are intersection points of projected silhouette curves and whose *arcs* are portions of projected silhouette curves. Arcs of the visibility map will be denoted by a, and silhouette curves by e. The curve whose projection contains the arc a is denoted $e(a)$. Note that a single silhouette curve can induce more than one arc, so for two arcs a, a' we can have $e(a) = e(a')$. It will be convenient to also consider the projections of visible endpoints of silhouette curves (that is, visible vertices) as nodes, as indicated in Fig. 2. Since we cut $\sigma(o)$ into two pieces when it changes direction with respect to the x-axis, the arcs of $\mathcal{M}(\mathcal{P})$ are x-monotone.

Curtains. For a curve e in $\mathbb{R}^3$ define the *curtain* of e, denoted curt(e), as the ruled surface constructed by taking a vertical ray pointing downward and moving its starting point from one end of e to the other. Thus, if e is a segment then curt(e) is an infinite polygon defined by e and two unbounded edges, each parallel to the z-axis. For a set E of curves we let curt(E) := {curt(e) | $e \in E$}.

Computing visibility maps. Our algorithm is based on the existing output-sensitive hidden-surface removal algorithm from [3]. Hence, we give a brief overview of this algorithm.

The algorithm is a plane-sweep algorithm. It sweeps over the viewing plane from left to right, detecting the arcs of the visibility map along the way. There are two types of event points: projections of object vertices (these are known in advance), and nodes of the visibility map (these will be computed as the sweep progresses).

When the projection of an object vertex v is reached by the sweep line, the algorithm checks whether v is visible. This is done by shooting a ray from v vertically upward. The vertex v is visible if and only if no object is hit by the ray. (Thus the algorithm needs a supporting data structure that can answer vertical ray shooting queries.) If v is visible, its projection is a node of the visibility map. This node will then be treated as an event for the sweep, as described next.

When a node of the visibility map is reached by the sweep line, the algorithm proceeds as follows. First the arcs ending at that node—this information can easily be maintained—are reported. Next it is determined whether any new arcs start at the node, that is, whether any arcs have the node as their left endpoint. This can be decided based on the two silhouette curves defining the node. For each new arc a, its right endpoint is computed and inserted into the event queue.

It remains to explain how to compute the right endpoint of a given arc a of the visibility map. An arc a can end for two reasons. One is that the silhouette curve $e(a)$ defining a ends. The other is that proj($e(a)$) intersects some other projected silhouette curve proj(e') such that either $e(a)$ becomes invisible or e' becomes visible—see Fig. 3. This can be detected by ray shooting in a set of curtains, as described next. When $e(a)$ becomes invisible because it disappears below some object o,

[3] For simplicity of presentation we assume o does not have any vertical facets, so that $\sigma(o)$ is uniquely defined. It is easy to adapt the definitions to the general case.

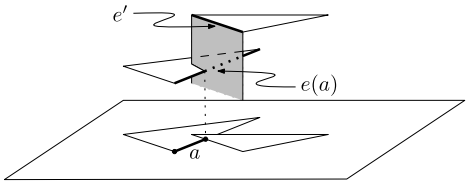

Fig. 3. The endpoint of arc a is the intersection of proj($e(a)$) and proj(e'), and it corresponds to the ray along $e(a)$ hitting curt(e'). (Note that the objects pictured here are not fat, but could be the top surfaces of fat polyhedra. We draw the objects in this way to ease visualization.)

then the ray along $e(a)$ must hit the curtain hanging from one of o's silhouette curves. When some other silhouette curve e' becomes visible, something similar holds. To this end, we define a ray $\rho(a)$ for an arc a of the visibility map as follows. Let q be the point on $e(a)$ projecting onto the left endpoint of a. Project the portion of $e(a)$ to the right of q onto the object $o(q)$ immediately below q. (If there is no such object, we project onto a plane below all objects.) This gives us a ray on the surface of $o(q)$ whose projection contains a. Note that a ray can be curved and can contain multiple segments. It can be argued [3] that the point where $\rho(a)$ hits curt(e') corresponds to the point where the silhouette curve e' becomes visible. Since any curtain hit by the ray along $e(a)$ is also hit by $\rho(a)$—after all, $\rho(a)$ is below $e(a)$—we can detect events where $e(a)$ becomes invisible by shooting along $\rho(a)$ as well.

The next lemma summarizes the discussion above.

Lemma 2. *(See [3].) Let E be the set of silhouette curves of the objects in $\mathcal{P}$. The right endpoint of an arc a of $\mathcal{M}(\mathcal{P})$ is the leftmost of the following event points*:

- *The projection of the right endpoint of $e(a)$.*
- *The projection of the first intersection of $\rho(a)$ with a curtain in* curt(E).

3. The algorithm

As mentioned in the introduction, it seems hard to implement a structure for ray shooting in curtains that profits from the fact that the objects are fat. Therefore we use the following idea.

Consider a collection of curtains hanging from the silhouette curves of some set of objects that are all above a plane h. Now suppose we want to do ray shooting in those curtains with a query ray $\rho(a)$ that is below h. Then we can project all objects and the ray onto h, and shoot with the projected ray in the union of the projected objects; the point where the ray hits a curtain then corresponds to the point where the projected ray hits the union. This is true because in our application the ray will always be visible, so the projected ray cannot start inside the union. Unfortunately two-dimensional ray shooting is still too costly. If, however, we have to answer many queries, then we can project all of them onto h, and perform a sweep to detect when they intersect the union. Of course there will not be a plane h that nicely separates all objects from all rays. Therefore we construct a binary space partition[4] (a *BSP*, for short) on the objects. This will basically give us a collection of $O(\log n)$ planes that together separate any ray from all the objects. The ray will then be traced on each of these planes. Below, we make this idea more precise.

We start by describing the BSP in Section 3.1, then discuss in Section 3.2 the correspondence between ray shooting in curtains and tracing rays on a suitable set of planes, and finally we give the details of the algorithm in Section 3.3.

3.1. The data structure

A *balanced aspect ratio tree* (or *BAR-tree* for short) is a special type of BSP for storing points. It was introduced by Duncan [12,13]. The variant known as the *object BAR-tree* [9] stores objects rather than points and has proved especially useful in designing data structures for fat objects. It has been used as a basis for vertical ray shooting [4,7] as well as approximate range searching and nearest neighbor searching [9].

We denote the region associated with a node v in the object BAR-tree for $\mathcal{P}$ by region(v), and we let $\mathcal{P}_v$ denote the set of all objects $o \in \mathcal{P}$ intersecting region(v), clipped to region(v). The following lemma states the properties of the object BAR-tree we will need.

Lemma 3. *(See [9].) Let $\mathcal{P}$ be a set of n β-fat disjoint convex objects in $\mathbb{R}^d$. An object BAR-tree on $\mathcal{P}$ is a BSP tree $\mathcal{T}$ for $\mathcal{P}$ with the following properties*:

[4] A BSP is a recursive partition of space by hyperplanes that continues until each subspace contains a constant number of input objects. The recursive construction can be modeled as a binary tree $\mathcal{T}$, called a BSP tree, where each node of $\mathcal{T}$ stores one of the splitting planes in the BSP. See Chapter 12 of [6] for a more extensive overview.

(i) *the tree has $O(n)$ leaves and each leaf region intersects $O(1/\beta)$ objects from $\mathcal{P}$;*
(ii) *the depth of the tree is $O(\log n)$;*
(iii) *for each node ν,* region(ν) *has constant complexity and fatness.*

De Berg [4] has shown how to augment an object BAR-tree $\mathcal{T}$ in $\mathbb{R}^3$ with secondary structures, so that vertical ray shooting can be performed efficiently. The augmentation is as follows.

- For each leaf node μ of $\mathcal{T}$, we store the set $\mathcal{P}_\mu$ in a list $\mathcal{L}_\mu$.
- For an internal node ν, let h_ν denote the splitting plane stored at ν.
  - If h_ν is vertical, then we store the set $\{h_\nu \cap o\colon\ o \in \mathcal{P}_\nu\}$—that is, the cross-sections of the polyhedra in $\mathcal{P}_\nu$ with h_ν—in a structure $\mathcal{T}_\nu$, which is an optimal point-location structure [17] on the trapezoidal map defined by $h_\nu \cap \mathcal{P}_\nu$.
  - If h_ν is not vertical, then ν has two associated data structures, $\mathcal{T}_\nu^+$ and $\mathcal{T}_\nu^-$, defined as follows.
    Let $\mathcal{P}_\nu^+$ denote the set of object parts from $\mathcal{P}_\nu$ lying above h_ν. Thus $\mathcal{P}_\nu^+ = \mathcal{P}_\mu$, where μ is the child of ν corresponding to the region above h_ν. Let proj($\mathcal{P}_\nu^+$) denote the set of vertical projections of the objects in $\mathcal{P}_\nu^+$ onto h_ν. Then $\mathcal{T}_\nu^+$ is an optimal point-location structure for $U(\text{proj}(\mathcal{P}_\nu^+))$, the union of proj($\mathcal{P}_\nu^+$). In our application, we not only store the point-location structure for $U(\text{proj}(\mathcal{P}_\nu^+))$, but also an explicit list of all union edges.
    The associated structure $\mathcal{T}_\nu^-$ is defined similarly, but this time for the object parts below h_ν.

Lemma 4. *(See [4].) The data structure described above requires $O((\frac{1}{\beta^5}\log^2\frac{1}{\beta})n\log^3 n(\log\log n)^2)$ storage and $O((\frac{1}{\beta^5}\log^2\frac{1}{\beta})n\times\log^4 n(\log\log n)^2)$ preprocessing time for constant-complexity convex β-fat polyhedral objects, and $O(\frac{1}{\beta^{14}}n\log^{7+\varepsilon} n)$ storage and $O(\frac{1}{\beta^{14}}n\log^{8+\varepsilon} n)$ preprocessing time for constant-complexity convex β-fat curved objects. With this structure, we can answer vertical ray shooting queries in $O(\log^2 n + 1/\beta)$ time.*

Recall that we want to use the structure not only to answer vertical ray-shooting queries in the given set of objects, we also want to use it for ray shooting in the curtains hanging from the objects' silhouette curves. The idea is as follows. Suppose that the query ray ρ is located inside the region of some leaf μ. Then any object above ρ (except for the $O(1/\beta)$ objects stored at μ) will be separated from ρ by some of the splitting planes stored at nodes on the path to μ. Hence, the ray shooting query can be answered by tracing ρ in the unions stored at these nodes.

There is one problem with this approach, however. The query rays are along the projections of (parts of) silhouette curves onto the object immediately below them. These objects and, hence, the query rays can be cut into many pieces by the BSP.[5] At the points where a ray is cut into pieces, it moves to a different leaf region. Then we would have to trace the ray on a different set of planes, because the path from the root changes—something we cannot afford to do too often.

To avoid this problem we proceed as follows. Let $\partial_{\text{top}}(o)$ denote the top surface of an object o, that is, the part of the boundary of o visible from above. For each object $o \in \mathcal{P}$ we will store the union of the projection of a certain subset $\mathcal{P}(o) \subset \mathcal{P}$ onto $\partial_{\text{top}}(o)$. The subset $\mathcal{P}(o)$ is defined as follows. Call an object o *large* at a node ν of $\mathcal{T}$ if o intersects region(ν) and the following two conditions are met: (i) size(o) $<$ size(region(parent(ν))) and (ii) either size(o) $\geqslant$ size(region(ν)) or ν is a leaf. Now we define

$$\mathcal{P}(o) := \{o' \in \mathcal{P}\colon \text{there is a node } \nu \text{ such that } o \text{ is large at } \nu,\ o' \text{ intersects region}(\nu), \text{ and } o' \text{ is above } o\}.$$

Finally, we also store the union of the projections of all the objects in $\mathcal{P}$ onto the xy-plane. (The xy-plane can be seen as a dummy object added below the whole scene, which is large at the root of $\mathcal{T}$.)

Next we analyze the cost of the additional information. We need the following lemma.

Lemma 5. *Any object $o \in \mathcal{P}$ is large at $O(\log n)$ nodes, and at any node ν there are $O(1/\beta)$ large objects.*

Proof. By Lemma 3(iii) we know that every cell of $\mathcal{T}$ is $O(1)$-fat. Lemma 1 then implies that any collection of disjoint cells has density $O(1)$. Therefore, since the cells at any level of the BAR-tree are disjoint, the number of nodes ν in any level of the BAR-tree intersecting some $o \in \mathcal{P}$ with size(region(ν)) $\geqslant$ size(o) is $O(1)$. An object o can only be large at the node ν if size(region(parent(ν))) $\geqslant$ size(o). Thus, the number of cells per level at which o can be large is $O(1)$. Finally we know that $\mathcal{T}$ has $O(\log n)$ levels by Lemma 3, proving the first part of the lemma.

To prove the second part of the lemma, we note that a set of disjoint β-fat objects has density $O(1/\beta)$ by Lemma 1. This immediately proves that there are only $O(1/\beta)$ large objects at any internal node. For leaf nodes this follows from Lemma 3(i). □

Using Lemma 5 we can prove a bound on the total size of all sets $\mathcal{P}(o)$.

[5] The fact that the objects may be cut into many pieces also prevents us from applying the following simple strategy: compute the object BAR-tree, use it to find a depth order on the resulting set of pieces, and apply the algorithm of Katz et al. [16]. The problem is that the visibility map of the pieces may be much more complex than the visibility map of the original objects.

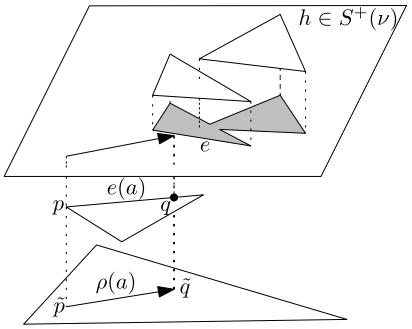

Fig. 4. $\rho(a)$ hits a curtain in curt(E) at point q when its projection intersects a silhouette curve of a union stored at $S^+(\nu)$.

Lemma 6. $\sum_o |\mathcal{P}(o)| = O((1/\beta) \cdot n \log n)$.

Proof. We have

$$\sum_o |\mathcal{P}(o)| \leqslant \sum_\nu \{(\#\text{ large objects at } \nu) \cdot (\#\text{ objects intersecting region}(\nu))\}$$
$$\leqslant O\left(\frac{1}{\beta}\right) \cdot \sum_\nu |\mathcal{P}_\nu| \leqslant O\left(\left(\frac{1}{\beta}\right) \cdot n \log n\right),$$

where the last inequality follows from [4]. □

Together with the known bounds on the union of fat objects [5,19] this is easily seen to imply that the total amount of storage and preprocessing time needed to construct the unions of the projections of $\mathcal{P}(o)$ onto the top surfaces $\partial_{\text{top}}(o)$ is upper bounded by the bounds in Lemma 4.

3.2. Tracing an arc

Recall that the right endpoint of an arc a can be found by shooting with $\rho(a)$ in curt(E). Next we explain how to find the right endpoint of a using the unions stored in $\mathcal{T}$ and the unions on the objects' top surfaces. The key is to find a collection of $O(\log n)$ unions such that the first point where $\rho(a)$ hits a curtain corresponds to the first point where one of the unions is hit.

To this end we first define for a node ν a collection $S^+(\nu)$ of $O(\log n)$ splitting planes:

$$S^+(\nu) := \{\text{splitting planes } h_{\nu'} \colon \nu' \text{ is an ancestor of } \nu \text{ and region}(\nu) \text{ is below } h_{\nu'}\}.$$

Let $e(a)$ be the silhouette curve defining an arc a, and let $p \in e(a)$ be the point projecting onto the left endpoint of a. Recall that $\rho(a)$ is a ray on the top surface of the object o directly below p. We denote the projection of p onto o by $\tilde{p}$. The first curtain hit by $\rho(a)$ can now be found using the following lemma.

Lemma 7. *Let $\rho(a)$ be a ray on $\partial_{\text{top}}(o)$ and let $\tilde{p}$ be the starting point of $\rho(a)$. Let ν be the node in $\mathcal{T}$ such that $\tilde{p} \in$ region(ν) and o is large at ν. Then the first curtain from* curt(E) *inside* region(ν) *hit by $\rho(a)$ corresponds to the first of the following events*:

(i) *$\rho(a)$ hits the union of the projection of the objects in $\mathcal{P}(o)$ onto $\partial_{\text{top}}(o)$;*
(ii) *the projection of $\rho(a)$ onto $h_{\nu'}$ hits the union stored on $h_{\nu'}$, for some $\nu' \in S^+(\nu)$.*

Proof. Note that the node ν referred to in the lemma is unique and must exist, since we consider the xy-plane to be a dummy object below the whole scene.

Let $\tilde{q}$ be the first point where $\rho(a)$ intersects a curtain in curt(E), let e be the silhouette curve defining the curtain, and let $q \in e$ be the point directly above $\tilde{q}$. If $q \in$ region(ν) then the object containing the silhouette curve e is a member of $\mathcal{P}(o)$ and we are in case (i). Otherwise there is a splitting plane $h_{\nu'}$ stored at some ancestor ν' of ν with q above $h_{\nu'}$ and $\tilde{q}$ below $h_{\nu'}$. Then the relevant portion of e must be part of the union stored at the first such node ν' (as seen from the root of $\mathcal{T}$). See Fig. 4.

Conversely, since all the unions considered are generated by (portions of) objects above o, we know that $\rho(a)$ cannot hit such a union before it hits a curtain. □

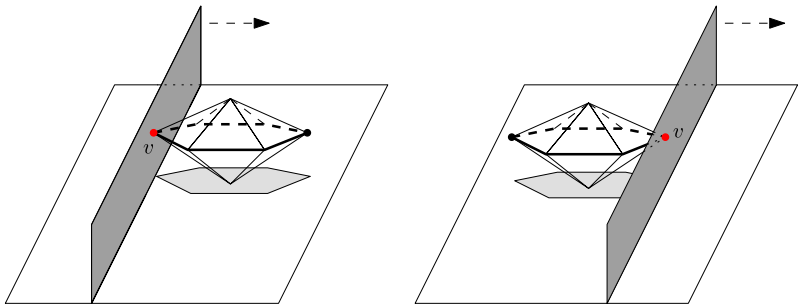

Fig. 5. Cases (iii) and (iv).

3.3. Details of the algorithm

We now describe our algorithm for computing the visibility map of a set $\mathcal{P} = \{o_1, \ldots, o_n\}$ of convex, constant-complexity, β-fat objects. The algorithm is a space-sweep algorithm that moves a sweep plane h parallel to the yz-plane from left to right through space. The space sweep induces a plane sweep for each of the unions stored in $\mathcal{T}$. Thus, instead of thinking about the algorithm as a 3D sweep, one may also think about it as a number of coordinated 2D sweeps. That is, while we sweep $\mathbb{R}^3$ with h, we also sweep each (non-vertical) splitting plane h_ν with the line $h \cap h_\nu$. Such a 2D sweep is performed to detect intersections of the union on h_ν with certain rays (projected onto h_ν). The same holds for the unions stored for each object: while we sweep $\mathbb{R}^3$ with h, we sweep the top surface $\partial_{\text{top}}(o)$ of each object o with the curve $h \cap \partial_{\text{top}}(o)$. Finally, the sweep of h induces a sweep on the viewing plane. As in the algorithm from [3], the visibility map will be computed as we go, so that at the end of the sweep the entire visibility map has been computed.

The space-sweep algorithm is supported by the following data structures:

- There is a global event queue Q, where the priority of an event is its x-coordinate. Initially, all vertices of the objects (that is, all endpoints of silhouette curves) are placed into Q. In addition, all vertices of any of the unions stored in $\mathcal{T}$ are placed into Q. During the sweep, new event points will be inserted into Q, for example endpoints of arcs of the visibility map. It is also possible that events will be removed before they are handled.
- For every splitting plane h_ν (and the top surface of every object o) we maintain a balanced binary tree, which we will call the *intersection–detection data structure*. This tree will store the edges of the union on the splitting plane (resp. $\partial_{\text{top}}(o)$) that intersect the sweep line $h \cap h_\nu$ (resp. $h \cap \partial_{\text{top}}(o)$) as well as the rays traced on it that intersect the sweep line; the edges and rays are stored in order of their intersection with the sweep line. Thus we are essentially running the standard line-segment intersection algorithm of Bentley and Ottmann [2] on the union edges and rays.

Next we discuss the events that can take place, and how they are handled. The first two events are essentially subroutines that we use in the other events.

(i) *The sweep reaches the left endpoint of an arc a.*
Let $e(a)$ be the silhouette curve defining a, and let $p \in e(a)$ be the point whose projection is the left endpoint of a. Let o be the first object that a vertical ray downward from p hits, and let $\tilde{p} \in o$ be the point where o is hit. Finally, let ν be the node where o is large such that $\tilde{p} \in \text{region}(\nu)$. Determine $S^+(\nu)$, and insert the portion of $e(a)$ starting at p into each of the intersection–detection data structures associated with the splitting planes in $S^+(\nu)$. (More precisely, the projection of the silhouette curve onto the plane is added.) Also add the projection of the silhouette curve onto $\partial_{\text{top}}(o)$ to the intersection–detection structure for o. Determine any new events using these data structures in the standard way (that is, by checking new pairs of adjacent elements); add any new events to Q. Finally, add the following three events to Q: the right endpoint of $e(a)$, the (first) intersection of $\rho(a)$ with the boundary of region(ν), and the (first) intersection of $\rho(a)$ with the silhouette of o.

(ii) *The sweep reaches the right endpoint of an arc a.*
Determine ν and o as above. Remove a from all intersection–detection data structures in $S^+(\nu)$ and the intersection–detection data structure associated with o. Remove all events associated with a from Q. Check for new events in each of the intersection–detection data structures; add any new events to Q. Output a as an arc of $\mathcal{M}$. (Note that the right endpoint of an arc may be the left endpoint of one or two other arcs; in this case the left endpoints will be separate events, which are handled according to case (i).)

(iii) *The sweep reaches the left vertex v of a silhouette curve.*
(In other words, we reach the leftmost point of an object $o \in \mathcal{P}$.) Determine if v is visible by shooting a ray vertically up from it. If v is visible, two arcs start at the projection of v onto the viewing plane. Run the actions from case (i) for each of these arcs (see Fig. 5).

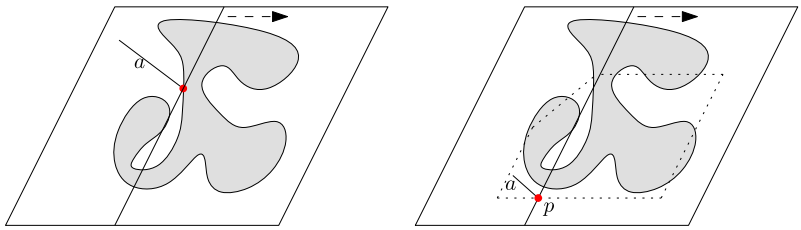

Fig. 6. Cases (v) and (vi).

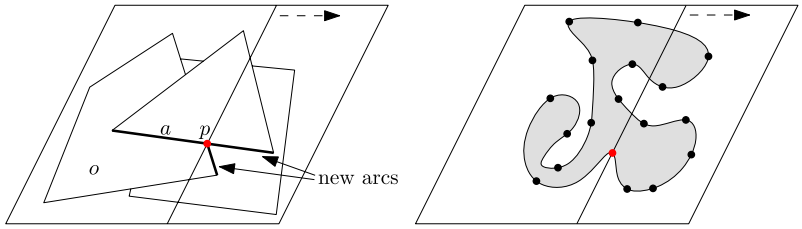

Fig. 7. Cases (vii) and (viii).

(iv) *The sweep reaches the right vertex v of a silhouette curve it is currently tracing.* (*Such a curve defines an arc currently intersected by the sweep line.*)
Run the actions from case (ii) for the arc ending at the projection of v.

(v) *The sweep reaches the intersection point of the union boundary on some splitting plane* (*or top surface of an object*) *and an arc a traced on the plane* (*or top surface*).
This case corresponds to a hitting a curtain in $\mathrm{curt}(E)$. Now the arc a ends. Run the actions from case (ii) for a. One or two new arcs may start at this point, at most one along the silhouette curve $e(a)$, and one along the silhouette curve corresponding to the curtain that is hit. Run the action from case (i) for the new arc(s) (see Fig. 6).

(vi) *The sweep reaches a point p where the projection of a currently visible silhouette curve onto the object o below hits the boundary of a cell ν where o is large.*
Let a be the arc defined by the silhouette curve. Remove a from all the intersection–detection data structures in $S^+(\nu)$ and all events associated with a from Q. Run the action for case (i) for the continuation of a. (The only thing that happens here is that the set $S^+(\cdot)$ changes, because the ray that we are tracing moves out of a cell where the object o on which the ray is traced is large.)

(vii) *The sweep reaches the point where the object o immediately below a currently visible silhouette curve changes.*
This can be detected because the visible silhouette curve is traced on $\partial_{\mathrm{top}}(o)$, and therefore we also know where it reaches the boundary of the top surface. Note that the projection of the point p where the curve reaches the boundary of the top surface is the right endpoint of an arc a. Run the actions from case (ii) for a. At most two new arcs start at p, one that is the continuation of a, and one that is along a silhouette curve of o (which became visible or stops being visible). Run the actions for case (i) on these curve(s) (see Fig. 7).

(viii) *The sweep reaches a point on a splitting plane* (*or top surface of an object*), *where a union edge starts or ends.*
In this case we only have to update the relevant intersection–detection data structure, check for new events in the intersection–detection data structures, and add any new events to Q.

Lemma 8. *The number of events of types* (i)–(vii) *is $O(n + k\log n)$, where k is the complexity of $\mathcal{M}$, and the total number of events of type* (viii) *is $O((\frac{1}{\beta^5}\log^2\frac{1}{\beta})n\log^3 n(\log\log n)^2)$ for constant-complexity fat polyhedra and $O(\frac{1}{\beta^{14}}n\log^{7+\varepsilon} n)$ for constant-complexity fat curved objects.*

Proof. Clearly, the number of events of types (i), (ii), (iv), (v), and (vii) is $O(k)$, since they can be charged to a node of $\mathcal{M}$. The number of events of type (iii) is $O(n)$. It remains to bound the number of events of type (vi). Consider the portion of a silhouette curve $e(a)$ defining some arc a. This portion has a unique object o immediately below it. Since o is large at $O(\log n)$ cells by Lemma 5 and the projection of $e(a)$ onto o can leave any cell only a constant number of times, we can conclude that there are only $O(\log n)$ type (vi) events for any arc a, this giving $O(k\log n)$ such events in total.

The bound on the number of events of type (viii) follows from Lemma 4. □

Lemma 9. *The time taken for each event of types* (i)–(vii) *is $O(\log^2 n)$, and the time taken for each event of type* (viii) *is $O(\log n)$.*

Proof. In all event types, we may need to perform several actions: vertical ray shooting, updating intersection–detection data structures, determining a set $S^+(\nu)$, and updating Q.

By Lemma 4, the time taken for the vertical ray shooting is $O(\log^2 n)$. Each event needs to do only a constant number of ray shooting queries, so this is $O(\log^2 n)$ in total. The intersection–detection data structures are balanced binary trees, so updates take $O(\log n)$ time. At each event we have to update $O(\log n)$ intersection–detection data structures, so the total time taken for updating is $O(\log^2 n)$. Determining new events in the intersection–detection data structures takes $O(1)$ per data structure, so the total amount of time taken for events of type (iii) is $O(\log^2 n)$. Determining a set $S^+(\nu)$ can be done in $O(\log n)$ time by searching in $\mathcal{T}$. At each event we may have to remove $O(\log n)$ event points from Q, each removal taking $O(\log n)$ time. Hence, all events of types (i)–(vii) can be handled in $O(\log^2 n)$ time, as claimed.

The events of type (viii) require only $O(\log n)$ time, since they only involve a constant number of operations on a single intersection–detection data structure. □

The correctness of the algorithm follows from Lemmas 2 and 7 as well as the correctness of the algorithm in [3]. We conclude with the following theorem.

Theorem 1. *The visibility map of a set of n disjoint constant-complexity convex β-fat polyhedra in $\mathbb{R}^3$ can be computed in time $O((\frac{1}{\beta^5}\log^2\frac{1}{\beta})(n\log n(\log\log n)^2+k)\log^3 n)$, where k is the complexity of the visibility map. When the objects are curved (and disjoint, constant-complexity, convex, and β-fat) the visibility map can be computed in time $O(\frac{1}{\beta^{14}}(n\log^{5+\varepsilon} n+k)\log^3 n)$.*

4. Conclusion

We presented the first algorithm to compute the visibility map of a set of fat convex objects that does not need a depth order and that runs in $O((n+k)\,\text{polylog}\,n)$ time.

One obvious open problem is to further reduce the running time, either by getting rid of some logarithmic factors or by reducing the dependency on the fatness factor β (which is currently quite bad). A second open problem is to extend the results to non-convex objects. Finally, it would be interesting to find an approach that works for low-density scenes, and not just for fat objects. The main problem here is that the union complexity of the projection of a low density scene can be $\Omega(n^2)$, so the approach would need to use a different data structure than the one presented in this paper.

References

[1] P.K. Agarwal, J. Matoušek, Ray shooting and parametric search, SIAM J. Comput. 22 (4) (1993) 794–806.
[2] J. Bentley, T. Ottmann, Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput. 28 (1979) 643–647.
[3] M. de Berg, Ray Shooting, Depth Orders and Hidden Surface Removal, Lecture Notes in Computer Science, vol. 703, Springer-Verlag, New York, 1993.
[4] M. de Berg, Vertical ray shooting for fat objects, in: Proc. 21st Annual Symposium on Computational Geometry, 2005, pp. 288–295.
[5] M. de Berg, Improved bounds on the union complexity of fat objects, Discrete Comput. Geom. 40 (1) (2008) 127–140.
[6] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, third ed., Springer-Verlag, Berlin, Germany, 2008.
[7] M. de Berg, C. Gray, Vertical ray shooting and computing depth orders for fat objects, SIAM J. Comput. 38 (1) (2008) 257–275.
[8] M. de Berg, M.H. Overmars, Hidden-surface removal for c-oriented polyhedra, Comput. Geom. Theory Appl. 1 (1992) 247–268.
[9] M. de Berg, M. Streppel, Approximate range searching using binary space partitions, Comput. Geom. Theory Appl. 33 (3) (2006) 139–151.
[10] M. de Berg, A.F. van der Stappen, J. Vleugels, M.J. Katz, Realistic input models for geometric algorithms, Algorithmica 34 (1) (2002) 81–97.
[11] M. Bern, Hidden surface removal for rectangles, J. Comput. System Sci. 40 (1990) 49–69.
[12] C. Duncan, Balanced aspect ratio trees, PhD thesis, Johns Hopkins University, 1999.
[13] C. Duncan, M. Goodrich, S. Kobourov, Balanced aspect ratio trees: Combining the advantages of k-d trees and octtrees, in: Proc. 10th Annual ACM-SIAM Sympos. on Discrete Algorithms, 1999, pp. 300–309.
[14] M.T. Goodrich, M.J. Atallah, M.H. Overmars, An input-size/output-size trade-off in the time-complexity of rectilinear hidden surface removal, in: Proc. 17th Int. Coll. on Automata, Languages and Programming, in: Lecture Notes in Computer Science, vol. 443, Springer-Verlag, 1990, pp. 689–702.
[15] R.H. Güting, T. Ottmann, New algorithms for special cases of the hidden line elimination problem, Comp. Vision Graphics Image Process. 40 (1987) 188–204.
[16] M.J. Katz, M. Overmars, M. Sharir, Efficient hidden surface removal for objects with small union size, Comput. Geom. 2 (4) (1992) 223–234.
[17] D. Kirkpatrick, Optimal search in planar subdivisions, SIAM J. Comput. 12 (1983) 28–35.
[18] M. McKenna, Worst-case optimal hidden surface removal, ACM Trans. Graphics 6 (1987) 19–28.
[19] J. Pach, G. Tardos, On the boundary complexity of the union of fat triangles, SIAM J. Comput. 31 (2002) 1745–1760.
[20] F.P. Preparata, J.S. Vitter, M. Yvinec, Computation of the axial view of a set of isothetic parallelepipeds, ACM Trans. Graphics 9 (1990) 278–300.
[21] J. Reif, S. Sen, An efficient out-sensitive hidden surface removal algorithm and its parallelization, in: Proc. 4th Annual Symposium on Computational Geometry, 1988, pp. 193–200.
[22] M. Sharir, M.H. Overmars, A simple method for output-sensitive hidden surface removal, ACM Trans. Graphics 11 (1992) 1–11.